Avoiding Accidental Performance Decisions

Dave Papworth

Bob Colwell

(Pentium Pro architects)

Intel Corp.

June, 1997

What we said in '95

- ♦ Performance in complex machines is...complicated
- ♦ Simulate, don't guess
- ♦ Surprises always work against you
- ♦ Must include
  - Real workloads
    - Real apps (database), real OS kernel
  - I/O, TLBs, memory, bus overhead
  - Heuristics fill in for mis-speculation
  - Bus & chipset
- ♦ Can't do performance until functionality is there
  - So performance work is chronically behind schedule

What we've learned since P6 intro

- We were right about doing performance projections
- We were right about the details (bus, TLB, mem)
- We were right about basing decisions on perf projections and simulations
- But we've learned a few things since then:

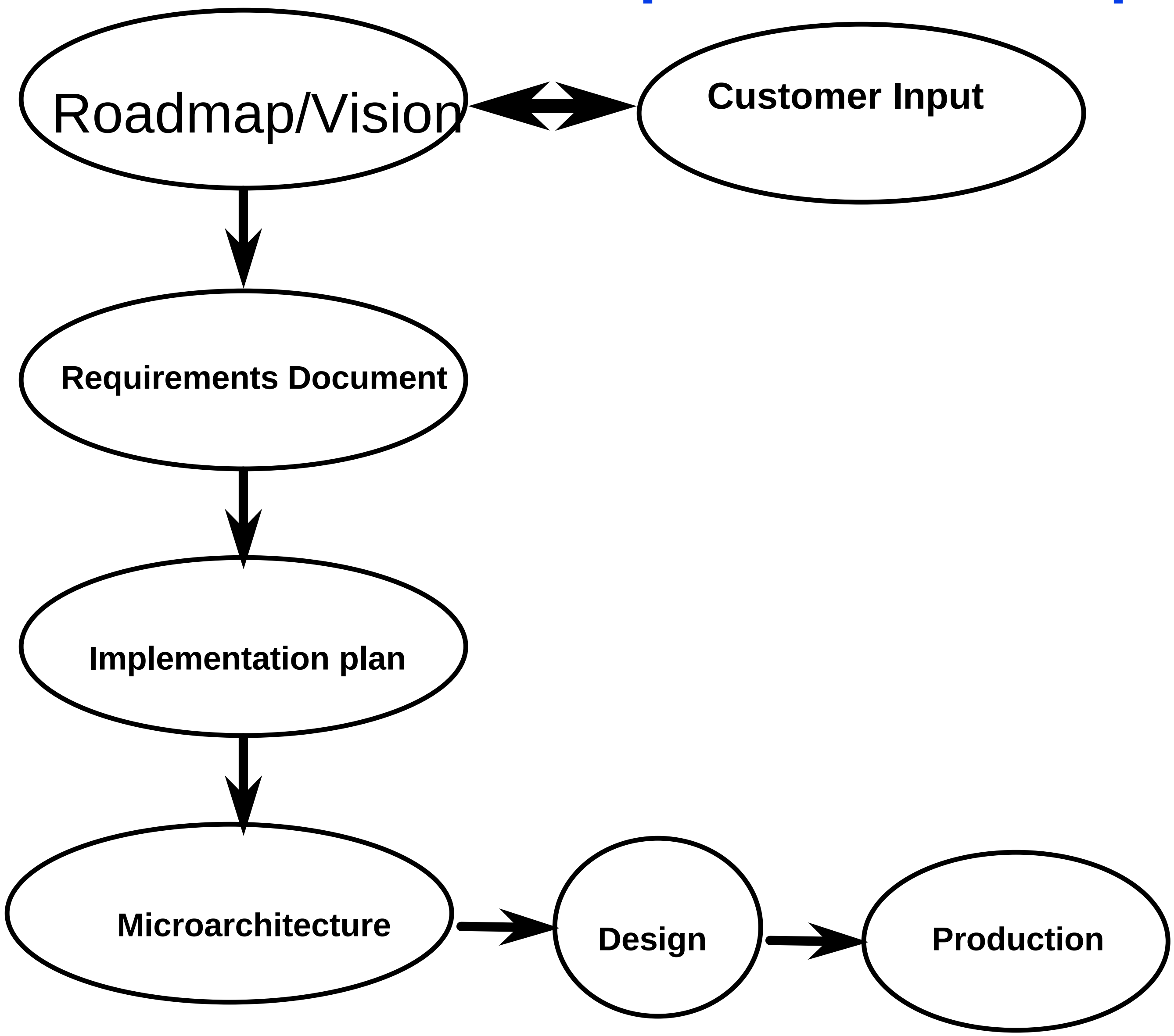
The decisions to watch are the ones you don't know you're making

Sources of "hidden decision making"

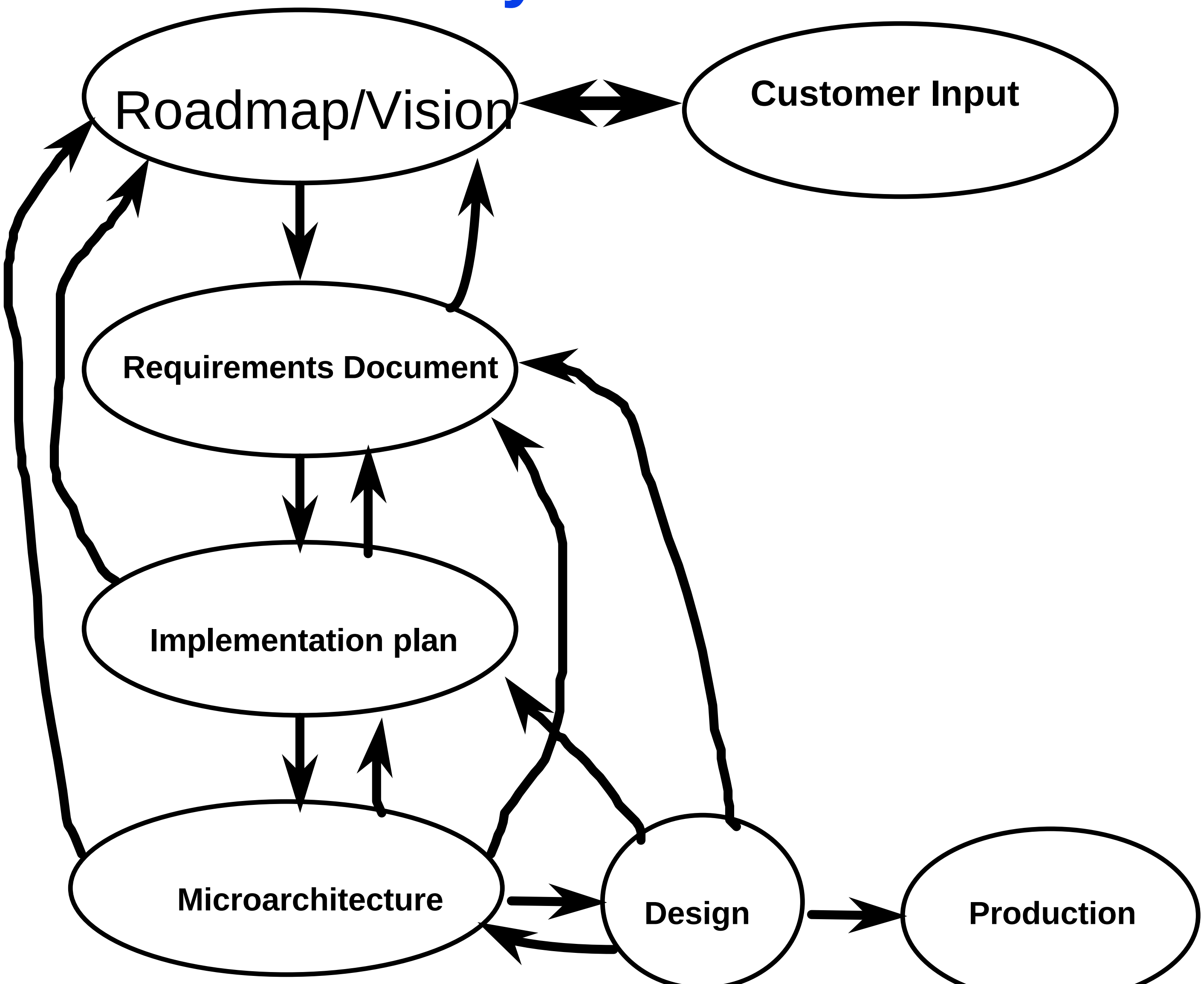
- Imperfections in performance prediction and planning
- Inadequate communication
- Large, long duration projects
- Human nature / Human dynamics

- Two common cases explored here
  - Product development becomes a one-way process
    - Beware "Product Requirements Documents"
  - Over-reliance on a few rules of thumb
    - Beware "too simple" simplifying assumption

Marketer's view of product development



How it really needs to work



Hazards of "top down" development

- Easier to add features than to remove them
- Performance and cost consequences of product feature sets hard to quantify in definition phase
- Development must proceed to a certain point before the necessary performance/cost/feature tradeoffs are fully evident. <u>Not a matter of successive refinement</u>
- What the product planner thinks:
  - This feature adds as much as 10% performance in some situations, and costs nothing the rest of the time
- What the product designer thinks:
  - This feature is a required part of the product, and any resulting performance cost just has to be accepted
- Beware of the "cookie jar" effect
  - Trying for too much, then backing off may give a suboptimal result for the portion you actually succeed at

Case study: Asynchronous PCI

- Feature:
  - Allow for 75MHZ memory bus AND 33 MHz PCI bus
  - Use fully asynchronous, independent clock domains
  - Optimize memory and I/O bus bandwidth
- Implication
  - I/O bridge has to do frequency domain translation
- Implementation decision
  - Synchronization/state machine added several clocks of latency to start and end of every PCI burst
  - Extra latency added even for synchronous case
- Environmental factors:
  - High performance networking (100Mbit+) did not yet exist
  - PCI controllers not optimized for long bursts
  - Multi-I/O, multi-memory, MP topology adds more latency

Case study: Asynchronous PCI

♦Actual results

- Extra latency obviated bandwidth benefits unless long bursts
- Most demanding app (Netbench) does not have long bursts
- Processor/memory bus never made it to 75 MHz
- Overhead of asynchronous feature impacted 66/33 MHz perf
- Initial Netbench performance was disappointing

♦Lessons:

- Feedback processes were lacking in design cycle
  - Feature not worth the performance cost of its implementation
  - Micro-benchmarks need to be established as “features”
- Product met literal spec, but not intent
- Result of trying to implement this feature (and failing) was worse than if we had decided up front to leave it out

The “Simplifying Assumption” Hazard

- “Small” decisions are often based on simple metrics
- Simple metrics may be based on false premises, or only applicable in limited contexts
- Hard to see effects of wrong decisions in early data
- Often too late to change when the real data arrives
- Watch out for “Slhadma’s Law”:

  “In computer system design, a sufficiently bad implementation of any one case may have a sizable impact on performance, even when that case is statistically infrequent.”

Example cases:

- Assumption:
  - Latency not a concern for I/O traffic (most I/O traffic is long bursts)
- Reality
  - Generally true for single agent transferring in one direction
  - Burst lengths get shorter as number of competing agents grows
  - Long latency worsens throughput for short bursts, causing more competition, causing even shorter bursts...
  - Existing cards optimized for existing systems (Triton), where there was no advantage to cache line oriented transactions
- Assumption:
  - Chipset can deliver 80% of theoretical bus bandwidth, there is bus bandwidth left over to allow for “instruction streaming”; a 4% perf win.
- Reality
  - True only for favorable address patterns
  - Typical MP bus-saturated applications have patterns which are not easily serviced at this rate
  - Actual result for streaming was about a 2% net loss

Example cases:

- Assumption:
  - Most graphics traffic is frame buffer writes, which can be accelerated by CPU write combining
- Reality
  - Popular benchmarks (Winstone97) spawned whole industry of graphics accelerator cards
  - Winstone compresses a “normal day’s work” into 10 minutes, placing very high weighting on (e.g.) changing cursor appearance
  - A fair amount of the resulting traffic is command strings going to the card, which must be strongly ordered and can only be write-combined with extra work by the card
  - Card vendors adapted to existing environment, and used cheapest possible solution (command fifos), which are order/combining-sensitive

Example cases:

- Assumption:
  - Implicit writebacks are rare transactions, and efficient handling is not important
- Reality
  - As database software has been tuned, less of the traffic is code fetches and sequential data reads, and more is “other stuff”
  - About 3% of the resulting bus transactions are implicit writebacks
  - The cost of the read/implicit writeback sequence was about 5 times as much as read/read due to the implementation
  - Slhadma’s law applies: implicit writeback represents 12 to 15% of the runtime even though only 3% of transactions

Dealing with the PRD hazard

- ♦ Must manage the human nature
  - In initial stages, projects invariably try for too much
- ♦ Top-down spec-driven design an unrealistic dream
  - Must allow for changes as implementation is refined
- ♦ Keep a close reign on marketing
  - Pick the absolute minimum feature set
  - Empower designers to question features
  - Make performance on micro-benchmarks part of spec

Managing simplifying assumptions

- ♦ Human nature; people focus on the common case
  - Keep track of performance on “uncommon” case
  - Test accuracy of data suggesting what is common and uncommon
- ♦ Create microbenchmarks
  - Measure many different axes throughout project
  - Understand performance on all cases; don’t let things “hide” behind mass of performance data in big benchmarks
- ♦ Pitch in and help measure performance (and make decisions) along the way
  - It can’t be spec’ed up front
  - It can’t be measured and retrofitted at the end